Détecteur de fautes pour le k-accord dans les systèmes inconnus et dynamiques

Élise Jeanneau, Thibault Rieutord, Luciana Arantes, Pierre Sens

Détecteur de fautes pour le k-accord dans les systèmes inconnus et dynamiques

Denis Jeanneau[1] [†] et Thibault Rieutord[2] [‡] et Luciana Arantes[1] et Pierre Sens[1]

[1] *Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, LIP6 - France*
[2] *INFRES, Telecom ParisTech - France*

Cet article définit le détecteur de fautes $\Pi\Sigma_{\perp,x,y}$ qui est suffisant pour résoudre le problème du k-accord, généralisation du consensus, à condition que $k \geq xy$. Le paramètre x est la force du détecteur à quorums et y est la force du détecteur de leader. Contrairement aux détecteurs de fautes existants, ce nouveau détecteur peut être implémenté dans un système inconnu et dynamique. L'article inclut un algorithme implémentant les propriétés de $\Pi\Sigma_{\perp,x,y}$.

Mots-clefs : Systèmes distribués, systèmes dynamiques, consensus, k-accord, détecteurs de fautes.

1 Introduction

Les algorithmes distribués considèrent traditionnellement des réseaux statiques et connus, modélisés par des graphes souvent connexes voire complets. Dans les systèmes modernes tels que les réseaux sans fil ou pair à pair, ces hypothèses ne sont pas réalistes.

Dans un réseau dynamique, les processus qui rejoignent ou quittent le système en cours d'exécution font partie du fonctionnement normal du système. Il est donc nécessaire de redéfinir la notion de faute : nous considérons que seuls les processus quittant le système de façon permanente sont fautifs. D'autre part, les canaux de communication peuvent également apparaître ou disparaître. Un système dynamique est souvent aussi un système inconnu : si certains processus rejoignent le système en cours d'exécution, il est impossible que tout les processus soient initialement connus.

Les solutions classiques de l'algorithmique distribuée ne sont pas toujours applicables telles quelles aux systèmes dynamiques. En effet, l'instabilité du système et l'absence d'informations initiales sur sa composition présentent de nouveaux défis qui nécessitent de nouveaux modèles et algorithmes.

Cet article s'intéresse au problème du k-accord, une généralisation du consensus. Le problème du k-accord nécessite que les processus du système proposent puis décident des valeurs en vérifiant les propriétés de validité (toute valeur décidée est une valeur proposée), d'accord (au plus k valeurs différentes sont décidées) et de terminaison (à terme, tous les processus corrects décident une valeur). Des solutions à ce problème existent pour des systèmes dynamiques, mais font l'hypothèse de communications synchrones [BRS+15].

Nous abordons le problème sous la perspective des détecteurs de fautes [CT96]. Il s'agit donc de définir les propriétés d'un détecteur de fautes qui soit suffisant pour résoudre le k-accord dans un système dynamique et inconnu. La suite de cet article définira un modèle de système (Section 2) ainsi que le détecteur $\Pi\Sigma_{\perp,x,y}$ (Section 3), puis proposera un algorithme implémentant ce dernier (Section 4).

[†] Denis Jeanneau est supporté par le Labex SMART dans le cadre du programme Investissements d'Avenir sous la référence ANR-11-LABX-65.
[‡] Thibault Rieutord est supporté par le projet ANR DISCMAT, dans le cadre de l'accord ANR-14-CE35-0010-01.

2 Modèle

Modélisation des processus. Le système est constitué d'un ensemble de n processus, dénoté Π. Ces processus sont *synchrones* (leurs vitesses relatives sont bornées). Les processus peuvent quitter ou rejoindre le système à tout moment (Π représente donc la liste de tous les processus qui participent au système à un moment donné). Un processus peut également tomber en panne ou récupérer d'une panne, mais un processus donné ne peut récupérer de pannes qu'un nombre fini de fois.

Le va et vient des processus faisant partie intégrante du fonctionnement normal d'un système dynamique, il convient de redéfinir la notion de processus fautif. Nous considérons un processus comme *fautif* si et seulement si ce processus quitte le système définitivement, ou tombe en panne et ne récupère jamais. Tout autre processus est *correct*, même s'il quitte et rejoint le système infiniment souvent ou tombe en panne et récupère. Nous dénotons $\mathcal{C}$ l'ensemble des processus corrects.

Chaque processus dispose d'une identité unique, mais ne connaît initialement que sa propre identité et doit communiquer avec les autres processus pour obtenir les leurs. Les processus ne connaissent pas la valeur de n ni $|\mathcal{C}|$ mais connaissent une valeur α telle que $\alpha \leq |\mathcal{C}|$.

Modélisation des communications. Les processus communiquent entre eux par *passage de messages*. Les communications sont *asynchrones* (le temps de transfert des messages n'est pas borné).

Les liens de communication entre les processus peuvent apparaître ou disparaître à tout moment. Lorsqu'un canal de communication est disponible, il est équitable : si un message est envoyé sur ce canal infiniment souvent, il est reçu infiniment souvent.

Du fait de la dynamicité des liens, le graphe de communication évolue au fil du temps. Dans le pire cas (aucun lien de communication n'est jamais disponible), aucun problème distribué ne peut être résolu. Il conviendra donc de définir les conditions de connectivité temporelles du graphe nécessaires au fonctionnement de nos algorithmes. Pour ce faire, le formalisme des Time-Varying Graphs [CFQS12] peut être utilisé. Intuitivement, chaque processus correct doit être capable de communiquer infiniment souvent avec un certain nombre d'autres processus corrects.

3 Détecteur de fautes

Un détecteur de fautes [CT96] est un oracle local fournissant aux processus des informations potentiellement non fiables sur les fautes dans le système. La force du détecteur dépend des garanties fournies sur la fiabilité de ces informations, et constitue une abstraction des hypothèses faites sur le modèle du système. Les détecteurs de fautes ont historiquement été utilisés pour contourner l'impossibilité de résoudre le consensus dans un système asynchrone en présence de fautes [FLP85, CHT96]. Dans ce papier, nous étendons la notion de détecteur de fautes afin de faire abstraction non seulement de l'asynchronie du système, mais aussi de sa dynamicité.

Dans les systèmes connus et statiques, le détecteur $\Pi\Sigma_{x,y}$ [MRS12] est suffisant pour résoudre le k-accord si $k \geq xy$. Ce détecteur fournit à chaque processus p_i un quorum de processus fiables (qr_i) et l'identité d'un leader partiel (dénoté $leader_i$). Le paramètre x représente la force de la propriété d'intersection portant sur la valeur de qr_i (plus la valeur de x est basse, plus la propriété d'intersection est forte). De la même façon, le paramètre y représente la force de la propriété de leader partiel portant sur $leader_i$.

Un leader partiel est défini de la façon suivante :

Definition 1 (Leader partiel)**.** *Un leader partiel p_l est un processus correct tel que, à terme, pour chaque processus p_i dont le quorum s'intersecte infiniment souvent avec celui de p_l, $leader_i = p_l$ pour toujours.*

Le leader partiel est une généralisation du leader Ω [CHT96]. La portée du leader dépend de l'intersection des quorums. Contrairement à Ω, il peut y avoir plusieurs leaders partiels dans un même système.

Dans ce papier nous présentons un nouveau détecteur adapté aux systèmes inconnus et dynamiques, $\Pi\Sigma_{\perp,x,y}$. Lorsque les informations disponibles ne permettent pas de fournir à p_i un quorum suffisamment fiable, la variable qr_i contiendra une valeur par défaut $\perp$. Les propriétés suivantes doivent être vérifiées :

— **Auto-inclusion** : chaque processus s'inclut lui-même dans chacun de ses quorums différents de $\perp$;
— **Vivacité** : à terme, pour tout p_i correct, qr_i est différent de $\perp$ et ne contient que des processus corrects ;

— **Intersection** : parmi tout ensemble de $x+1$ quorums différents de $\perp$, au moins deux s'intersectent ;
— **Connectivité** : chaque processus correct est capable de joindre infiniment souvent les processus dont le quorum s'intersecte infiniment souvent avec le sien ;
— **Leader partiel** : chaque processus correct peut être joint infiniment souvent par un leader partiel.

Ces propriétés définissent le détecteur $\Pi\Sigma_{\perp,x}$. Le détecteur $\Pi\Sigma_{\perp,x,y}$ est constitué de y instances de $\Pi\Sigma_{\perp,x}$ exécutées en parallèle. Il suffit alors qu'une seule de ces y instances vérifie la propriété de leader partiel.

Ce nouveau détecteur de fautes diffère de $\Pi\Sigma_{x,y}$ de plusieurs façons. Premièrement, la propriété de connectivité est ajoutée afin de s'abstraire de la dynamique du système. La propriété de leader partiel est également modifiée pour y ajouter une composante de connectivité : il ne suffit plus qu'un ou plusieurs leaders existent, il faut qu'ils puissent joindre les processus corrects. Ces nouvelles propriétés sont trivialement implémentées dans un système statique.

Deuxièmement, la valeur par défaut $\perp$ est introduite. Celle-ci permet au détecteur d'être implémenté dans un système inconnu. En effet, la propriété d'intersection doit s'appliquer dans le temps, ce qui signifie qu'un quorum retourné en fin d'exécution (et ne contenant donc que des processus corrects) doit s'intersecter avec un quorum retourné au début de l'exécution, même si le processus faisant appel au détecteur est fautif et n'a encore communiqué avec personne (et ne peut donc inclure que sa propre identité dans son quorum). La possibilité de retourner $\perp$ permet d'éviter cette contradiction.

Dans la version longue de cet article [JRAS16], nous prouvons que $\Pi\Sigma_{\perp,x,y}$ est suffisant pour résoudre le k-accord avec $k \geq xy$.

4 Algorithme

Cette section présente un algorithme permettant d'implémenter le détecteur $\Pi\Sigma_{\perp,x}$ et résume brièvement les hypothèses sur le système permettant de le prouver. Dans l'Algorithme 1, les processus émettent des requêtes datées par un numéro de ronde (lignes 8 et 20). A la réception d'une requête, chaque processus la rediffuse en y ajoutant son identité et l'identité de son candidat pour l'élection de leader (ligne 22). L'émetteur de la requête collecte ces réponses jusqu'à ce qu'il en ait reçues un nombre suffisant (ligne 12). Il peut alors utiliser les informations reçues pour définir son nouveau quorum (ligne 13) et sélectionner son leader (lignes 14 à 17). L'Algorithme 1 utilise les notations suivantes :

— **pos(**p, qr**)** : la position du processus p dans le quorum qr. Nous considérons que les processus dans un quorum sont totalement ordonnés par leur date d'ajout au quorum (et donc leur délai de réponse).
— **diffusion()** : une primitive de communication permettant à un processus d'envoyer un message à tous les processus se trouvant actuellement dans son voisinage immédiat, ainsi qu'à lui-même. Notez que cette primitive est très simple et n'inclut aucune garantie de remise (*best effort*).

Chaque message envoyé contient les valeurs suivantes :

— *src* : l'identité du processus émetteur de la requête ;
— *r_src* : le numéro de ronde du processus *src* correspondant à cette requête ;
— Q : l'ensemble des processus dont la réponse dont la réponse a été collectée dans ce message ;
— $candidats\_Q$: les candidats désignés par les processus dans Q.

Hypothèses. Afin d'assurer que l'Algorithme 1 implémente les propriétés de $\Pi\Sigma_{\perp,x}$, les trois hypothèses suivantes doivent être vérifiées. La première porte sur la connectivité du graphe de communication nécessaire au bon fonctionnement de l'algorithme. Les deux autres sont des hypothèses à motif de messages [MMR03].

1. Le système est un Time-Varying Graph [CFQS12] de classe 5-(α,γ) où α est la taille minimale d'un quorum et γ est le temps maximal mis par un processus pour recevoir ses propres messages. Cette hypothèse représente un degré de dynamicité du système que l'Algorithme 1 est capable de tolérer.
2. L'exécution suit un motif de messages de quorums gagnants généralisés. Intuitivement, cela signifie que chaque requête émise doit recevoir un certain nombre de réponses envoyées par un groupe de processus privilégiés. Cette hypothèse permet de vérifier la propriété d'intersection de $\Pi\Sigma_{\perp,x}$.
3. L'exécution contient une ou plusieurs γ-sources gagnantes à terme. Intuitivement, une γ-source gagnante à terme est un processus qui répond aux requêtes plus rapidement que les autres, et qui peut donc être identifié comme leader. Cette hypothèse permet de vérifier la propriété de leader partiel.

Algorithme 1 Implémentation de $\Pi\Sigma_{\perp,x}$ pour le processus p_i.

1: **initialisation**
2: $r_i \leftarrow 0$ // Le numéro de ronde local
3: $qr_i \leftarrow \perp$ // Le quorum retourné par le détecteur $\Pi\Sigma_{\perp,x}$ pour p_i
4: $tampon_i \leftarrow \{p_i\}$ // Le tampon utilisé pour construire le prochain quorum
5: $leader_i \leftarrow p_i$ // Le leader retourné par le détecteur $\Pi\Sigma_{\perp,x}$ pour p_i
6: $candidat_i \leftarrow \perp$ // Le candidat actuel de p_i pour l'élection de leader
7: $candidats\_tampon_i \leftarrow \emptyset$ // Les candidats désignés par les processus dans $tampon_i$
8: diffusion$(p_i, 0, \emptyset, \emptyset)$ // Diffusion de la requête initiale de p_i pour construire un premier quorum

9: **lors de la réception de** (src, r_src, Q, $candidats\_Q$) **envoyé par** p_j **faire**
10: **si** $src = p_i$ and $r\_src = r_i$ **alors** // Le message reçu est une réponse à la dernière requête de p_i
11: $tampon_i \leftarrow tampon_i \cup Q$; $candidats\_tampon_i \leftarrow candidats\_tampon_i \cup candidats\_Q$
12: **si** $|tampon_i| \geq \alpha$ **alors** // Suffisamment de réponses ont été reçues pour construire un nouveau quorum
13: $qr_i \leftarrow tampon_i$
14: $candidat_i \leftarrow p_l \mid ((pos(p_l, qr_i) = 1 \wedge p_l \neq p_i) \vee (pos(p_i, qr_i) = 1 \wedge pos(p_l, qr_i) = 2))$
15: // Le candidat de p_i est le processus (autre que lui-même) qui a répondu le plus vite
16: **si** $candidats\_tampon_i = \{p_i\}$ **ou** $candidats\_tampon_i = \emptyset$ **alors** $leader_i \leftarrow p_i$
17: **autrement** $leader_i \leftarrow candidat_i$
18: // p_i se sélectionne comme leader seulement si son quorum le désigne unanimement comme candidat
19: $tampon_i \leftarrow \{p_i\}$; $candidats\_tampon_i \leftarrow \emptyset$; $r_i \leftarrow r_i + 1$ // Initialisation de la prochaine ronde
20: diffusion$(p_i, r_i, \emptyset, \emptyset)$
21: **autrement si** $src \neq p_i$ **alors** // Le message reçu est une requête de src à laquelle p_i doit répondre
22: diffusion$(src, r_src, Q \cup \{p_i\}, candidats_Q \cup \{candidat_i\})$

La version longue de cette article [JRAS16] contient une description complète de ces hypothèses, ainsi qu'une preuve de la validité de l'Algorithme 1.

Afin d'implémenter $\Pi\Sigma_{\perp,x,y}$, il suffit d'exécuter y instances de l'Algorithme 1 en parallèle, de sorte que différentes exécutions peuvent permettre à un processus de choisir différents quorums et leaders.

Bilan. Les systèmes dynamiques et inconnus présentent de nouveaux défis pour la résolution des problèmes d'accord. Ces difficultés peuvent être résolues en modifiant la définition d'un détecteur de fautes afin de faire abstraction à la fois de l'asynchronie des communications et de la dynamicité du système.

L'algorithme présenté dans cet article montre qu'il est possible d'implémenter un tel détecteur de fautes dans un système dynamique et inconnu à l'aide d'hypothèses raisonnables. Avec quelques modifications, l'algorithme présenté dans [MRS12] peut alors être utilisé avec $\Pi\Sigma_{\perp,x,y}$ pour résoudre le k-accord.

Références

[BRS+15] Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. In *NETYS 2015*, pages 109–124, 2015.

[CFQS12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5) :387–408, 2012.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The Weakest Failure Detector for Solving Consensus. *JACM*, 43(4) :685–722, 1996.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 43(2) :225–267, 1996.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2) :374–382, 1985.

[JRAS16] Denis Jeanneau, Thibault Rieutord, Luciana Arantes, and Pierre Sens. Solving k-set agreement using failure detectors in unknown dynamic networks. *IEEE Transactions on Parallel and Distributed Systems*, 2016.

[MMR03] Achour Mostéfaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In *DSN*, pages 351–360, 2003.

[MRS12] Achour Mostéfaoui, Michel Raynal, and Julien Stainer. Chasing the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems. In *NCA 2012*, pages 44–51, 2012.